# A Reactive Architecture for Cloud-Based System Engineering

David Adjepon-Yamoah
Centre for Software Reliability
Newcastle University
Newcastle-upon-Tyne, United Kingdom
(d.e.adjepon-yamoah,

Alexander Romanovsky
Centre for Software Reliability
Newcastle University
Newcastle-upon-Tyne, United Kingdom
alexander.romanovsky,

Alexei Iliasov
Centre for Software Reliability
Newcastle University
Newcastle-upon-Tyne, United Kingdom
alexei.iliasov) @ncl.ac.uk

## ABSTRACT

The paper introduces an architecture to support system engineering on the cloud. It employs the main benefits of the clouds: scalability, parallelism, cost-effectiveness, multi-user access and flexibility. The architecture includes an open toolbox which provides tools as a service to support various phases of system engineering. The architecture uses the Open Services for Lifecycle Collaboration (OSLC) technology to create a reactive middleware that informs all stakeholders about any changes in the development artefacts. It facilitates the interoperability of tools and enables the workflow of tools to support complex engineering steps. Another component of the architecture is a shared repository of artefacts. All the artefacts generated during a system engineering process are stored in the repository, and can be accessed by relevant stakeholders. The shared repository also serves as a platform to support a protocol for formal model decomposition and group work on the decomposed models. Finally, the architecture includes components for ensuring the dependability of the system engineering process.

## Categories and Subject Descriptors

D.2 [**Software**]: Software Engineering; D.2.9 [**Management**]

## 1. INTRODUCTION

The complexity of the systems we are building now and will be building in the near future is the main challenge for the engineering community. Even though this increase in complexity is mainly driven by the market needs, there are other contributing factors such as regulations and standards. This calls for development and deployment of the advanced system and software processes that will assist all of the stakeholders in creating systems that are critical and dependable. These system processes will need to incorporate different steps, technologies, tools and services to meet the ever-changing needs of stakeholders.

Such advanced processes can be dramatically improved if they rely on an effective supporting architecture (infrastructure) that brings together and interconnects varying elements involved in system engineering. It will facilitate traceability, change management and the agile capturing and maintenance of requirements. This support will also help with tool integration and collaboration between developers, and between developers and stakeholders.

By recording the history of all system development and storing all development artefacts in a dedicated shared repository, such architecture will be able to support reactive system engineering when a change in any development artefact is propagated to all relevant parties (artefacts, tools and developers). This will allow the developers to react to this change and ensure traceability of all development artefacts and change management.

The repository of artefacts is the critical element of this architecture. There are now various types of shared repositories that support some phases of system engineering, such as SVN and GitHub. These repositories are mainly focused on the storing, accessibility and version control of the files that are kept in them. Our repository is part of a larger architecture that does not only support change management and traceability of system engineering artefacts, but links the artefacts with the tools/engineering steps in which they are either produced or used.

This short paper discusses our on-going work on developing such an architecture. We decided to use the cloud technology to support the architecture because of various benefits that it can bring. Hence, all the components of this architecture are being developed as a REST (Representational State Transfer) web service, which also ensures interoperability of these components. The architecture will be typically accessed by the stakeholders through a client plug-in into the individual IDEs (e.g. Eclipse-based) they use on their desktop computers. The overview of our architecture is shown in Figure 1.

## 2. BACKGROUND

Modern systems are getting increasingly complex with a mixture of products and services that are used in their development to meet the changing needs of businesses. This situation currently poses some challenges to system engineering. The development process has to adopt dynamic technologies

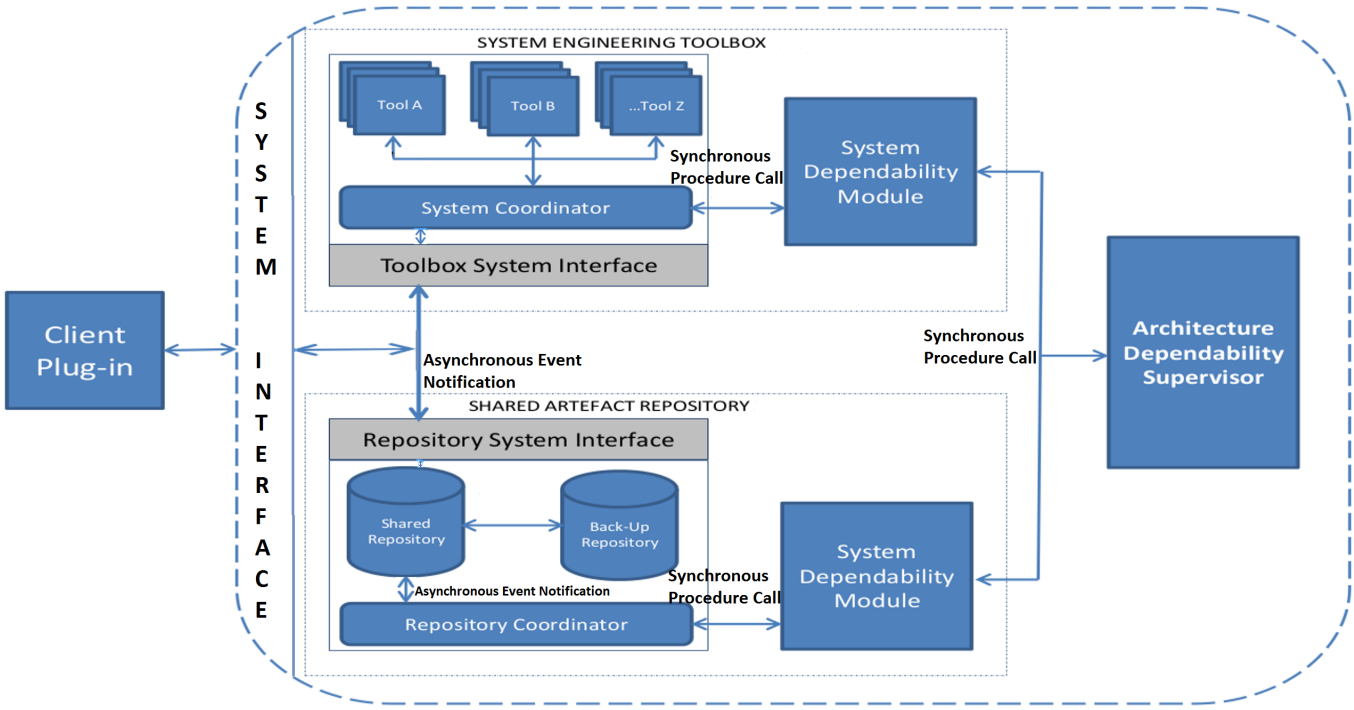


**Figure 1: High-Level Representation of the Cloud-Based Architecture**

and development methods to meet these needs. A framework that effectively supports the Open Services for Lifecycle Collaboration (OSLC) [1] specification can be used to address the problem of interoperability of products and services. OSLC is a new industry-driven standard for tool integration, that in particular supports requirements and change management. To further make the system development process dynamic and reactive, a shared repository of system engineering artefacts for various phases of system engineering is critical. Here, changes made to these artefacts could be sent as notifications to all stakeholders of the artefacts. The notification process could be supported through a publish/subscribe mechanism. The publish/subscribe paradigm is receiving increased attention for the loosely coupled form of interaction it provides in large scale settings [12]. In practice, the subscribers register their interests in a topic or a pattern of events and then receive events matching their interest. This idea could be used in system engineering to support change management and traceability of artefacts. In general, a dependable architecture that embodies the above introduced development methods and technologies for system engineering can appropriately accommodate the complexity of this process. Provisioning such an architecture on the cloud should be beneficial, as the cloud computing provides relatively cheap resources, multi-user access, global access, scalability, etc.

The OSLC technology and cloud computing are briefly discussed below. This section is completed by introducing some closely related architectures for system engineering.

## 2.1 OSLC

Open Services for Lifecycle Collaboration (OSLC) [1] is a global forum that develops standards and technologies for application life-cycle management. The motivation for its work is the fact that different organisations use different regulations and standards (management tools and data formats). This makes it difficult to exchange management data across the organisational boundary. Some obstacles in exchanging data may be the variety of data schema, computing platforms, management tools, and scalability matters. OSLC provides an interface for interoperability of tools and data formats. It addresses integration scenarios for a growing number of domains, including change management, test management, requirements management, automation, and performance monitoring. OSLC is important for IT organisations faced with a mix of system environments and application models (Cloud, Mobile, Social, Big Data Analytics, etc.). It can help to provide the capacity for more effective automation, communication and collaboration in system engineering. Here, different components of a system can implement different vendor tools that produce different formats of data and artefacts, and yet can function with no or negligible conflicts. Also, a workflow of tools can be created to support system engineering.

## 2.2 Cloud Computing

Cloud computing[11] is changing the current IT delivery model for services. The technologies working behind cloud computing are cluster computing, grid computing, peer-to-peer (P2P) computing, etc. The cloud platform offers scalability, flexibility, elasticity, on-demand resources utilisation, etc.

The industry and academia are now widely using the cloud platform for system development and research to gain its benefits. An interesting example though not system engineering related, is a joint project between IBM and Microsoft

[15] which aims to instil greater confidence in computations outsourced to the cloud. Pinocchio, a built system for efficiently verifying general computations while relying only on cryptographic assumptions is developed. With Pinocchio, the client creates a public evaluation key to describe the computation; this setup is proportional to evaluating the computation once.

There has been some work on supporting software development on the cloud. For example, the scalability of the cloud (Amazon Web Service - EC2) was used to test a complex set of test cases of the Google Chrome software [14]. The testing benefited from this as the large-scale software testing typically requires substantial computation and storage resources. Moreover, this work has greatly benifited from the use of the cloud as the testing process can be automated through a workflow requiring minimal human intervention. Another example is Microsoft Research project (Rise4Fun) [13] to support system verification using the Z3 SMT solver, on the cloud. Z3 exposes a number of APIs, but is in itself a low-level tool. The project provides various ways that lower the barrier of entry to using Z3. This includes using programming language abstractions and exposing Z3 as a web service on the cloud.

Many desktop applications are being migrated onto the cloud due to the benefits of cloud computing to support programming on the cloud. Examples of such IDEs are Eclipse Orion [18], Codenvy [3], CEclipse [20], etc. Cloud or Online IDE is based on the Browser/Server structure, and allows programmers to write their programs through the browser. Cloud IDEs provide the same basic features as the local IDEs, such as *code syntax highlighting*, and *compiling*. In addition they support collaborative development, easy setup with just a web browser and network connection, etc.

## 2.3 System Engineering Architectures

The two strands of relevant work are the Evidential Tool Bus [17] and the Open Framework for Software Engineering Tools (OPHELIA) [8].

With regards to the **Evidential Tool Bus** [17], the authors propose a tool combination of theorem provers, model checkers, static analyzers, test generators, etc. where many tools and methods are used in an ad-hoc combination within a single analysis. This sort of combination requires an integrating platform - a tool bus - to connect the various tools together; but the capabilities required go beyond those of platforms such as Eclipse. In the tool bus, all tools are co-equals, and are mainly coordinating components of the tool bus. The entities exchanged among clients of the bus - proofs, counterexamples, specifications, theorems, abstractions - have logical content, and the overall purpose of the bus is to gather and integrate evidence for verification or refutation.

The **OPHELIA** [8] platform provides a unified software engineering tools integration technology. The concept behind the project involves the definition of standardized set of interfaces abstracting functionalities of different kinds of software development tools. To maintain implementation language independence, CORBA technology is used to define the interfaces. As part of the Object Management Group (OMG) work, they have developed a comprehensive distributed open systems framework known as Common Object Request Broker Architecture (CORBA). CORBA is a standard for object middleware used in the heterogeneous environment. The main weakness of OPHELIA is of poor memory management, which is inherited from the CORBA platform. Also, the use of the CORBA platform as a middleware for integration introduces overhead costs through its use or brokers (translators).

# 3. ARCHITECTURE DESCRIPTION

The overview of the proposed architecture supporting system enginering on the cloud (see Figure 1) is now described briefly in terms of components, connectors, constraints, and dependability.

**Components:** The main sub-systems of this architecture are System and Component Interfaces, Tool and Repository Coordinators, Tool Box, and Shared Artefacts Repository. **System and Component Interfaces** provide globally unique names of handlers, based on internet domain names. This is generally specified as a unique Uniform Resource Identifier (URI). Also, access to system resources is regulated. All access requests that do not meet the defined standards are declined and noted. Provisions against malicious attacks are made. Again, the interface determines the capacity and ordering of resource requests for transactional management purposes. A standard of the relationships between components are described in the component interface(s). This defines the type, means and scope of transactions between components. The **Tool Coordinator** manages central updates of tools, defines the relationships between tools (workflow of tools), manages system artefacts manipulation processes (analysis, prediction, optimisation, scheduling, etc), and facilitates data presentation in different predefined (standard) formats. Also, the **Repository Coordinator** manages search and retrieval of artefacts. This coordinator ensures the support for reactive system engineering by notifying all relevant parties (artefacts, tools and developers) when there is a change in any development artefact. This process is made possible through a publish/subscribe mechanism. These coordinators (tool and repository) together ensure real-time monitoring, change management, traceability, capacity planning and fault diagnostics and tolerance. The **Tool Box** brings together various tools that support the system engineering phases. It also provides access to various versions of tools, provides fail-over support for tools, and facilitates communication between tools. The **Shared Artefacts Repository** stores system engineering artefacts (source code, test cases, models, patterns, documentations, requirements, etc.), it employs an efficient indexing process (to facilitate prompt access to artefacts), and it asynchronously backs its contents up in a remotely located repository.

**Connectors:** A connector defines the type of relationships between the components of a system. The connectors used in this architecture are **Asynchronous Event Notification** for external requests (from clients such as IDEs and Tools) to the system interface (API), **Peer-to-Peer Asynchronous Event Notification** (System Interface (API) to Repository Coordinator, Search and Retrieval requests to Repository Coordinator, and Shared Repository to Back-Up

Repository), and **Peer-to-Peer Synchronous Procedure Call** (Shared Repository to Back-Up Repository)

**Constraints:** Some constraints that have been experienced, and those that can be potentially experienced later are resources (time, budget, etc.), technology constraint, local standards (development, coding, etc.), public standards (HTTP, XML, XML Schema, WSDL, WADL, etc.), standard protocols, standard message formats, and skill profile of developer.

**Dependability:** Components such as the **Toolbox** system and the **Shared Artefacts Repository** system will have respective modules (**Architecture Dependability Module**) to ensure dependability. These modules will use availability and reliability metrics to assess the dependability of these two components. A supervisory dependability component (**Architecture Dependability Supervisor**) shown in Figure 1 collects availability and reliability data from the dependability modules, and compares them with those from the cloud platform metrics. Cloud service providers such as the Amazon Web Service, make dependability metrics available for all their users' virtual machines. These metrics are mainly classified as availability and reliability metrics [6]. The dependability data collected by the modules are obtained and sent synchronously as **Procedure Calls** to the dependability supervisor at defined time intervals. This process further reassures the dependability of the architecture.

The aim of our work is to design, implement and evaluate a cloud-based architecture (Figure 1) supporting system engineering. Its main novel characteristics are a middleware for reactive change management and traceability, a shared artefact repository supporting collaboration, an open system engineering toolbox ensuring tool interoperability, and a support to assess its dependability in run-time.

# 4. CURRENT WORK

This section reports the ongoing and recently completed work on developing several critical elements of the general architecture. The aim has been to conduct proof of concept designs and to explore possible solutions that will contribute to the architecture to be developed. We are using the two Eclipse IDEs in this work: the Rodin toolset for rigorous development of complex critical systems using Event-B [4] and the SafeCap IDE [10] developed to assist signalling engineers in designing complex stations and in ensuring their safety.

## 4.1 Reactive Middleware

The OSLC technology is being used in the architecture to create a reactive middleware that collect and interlink system engineering artefacts to support traceability and change management. We have created an OSLC adapter that runs as a cloud-based web service connected to Rodin using a plug-in designed to initiate the request for the project elements. The plug-in adds an embedded HTTP server (jetty) and implements, with the help of the Lyo framework and Apache Wink, a RESTful java servlet that serves Rodin projects and their parts. The developed prototype links requirements and formal models to the OSLC-based notification system. It exposes formal Event-B models up to the level of individual expressions and predicates to external tools (such as ProR[19] for requirement engineering). Every major element of a model has its unique identifier and is made available for other tools to inspect.

## 4.2 Model Checking Tool

In this project, a model-checker called ProB[9] is wrapped as a web service based on the REST protocol to become part of the architecture toolbox. A client plug-in created for the SafeCap IDE sends specifications of railway signalling models for verification on the developed web service. The web service is deployed on Tomcat Server 6.0. The main CRUD (Create, Read, Update and Delete) operation used is the POST to receive REQUESTS from client IDEs as new entries of data (JSON file) into the web service. Another operation, GET is used to facilitate a RESPONSE with POST to return the verification results to the client.

## 4.3 Theorem Provers on the Cloud

In this work we have created a theorem prover tool as part of our architecture toolbox and a plugin that connects the Rodin IDE to this tool. The tool is using the Why3 software [5] that brings together a collection of some well-known theorem provers (Alt-Ergo, Z3, Yices, Vampire, SPASS, etc.). The web service is hosted on the Amazon AWS cloud. The plugin maps Event-B mathematical language into the Why3 notation, the tool uses Why3 to implement subsequent translation into TPTP and SMT-LIB formats compatible with a wide range of existing provers.

In this work we have also created and experimented with a shared artefact repository as part of our architecture (Figure 1). The cloud-based prover service keeps a detailed record of all artefacts for every proof attempt in the shared artefact repository. These artefacts are mainly proof obligations, supporting lemmas and translation rules. Provisions are made to obfuscate sensitive proof obligations. The repository is a relational database service running on the Amazon AWS cloud [2]. Here, all queries from the prover service are made as HTTP requests using the repository's URI. We plan to perform some analysis of relative prover performance by mining this repository to identify the correlation between prover success, verification time, etc.

# 5. FUTURE WORK

Our future work will focus on creating an integrated environment out of the elements of the architecture being developed, and on evaluating it using realistic industry-driven case studies. Some other elements of the architecture that need to be developed are described below.

**Formal Model Decomposition Support**: The current state of methodological and tool support for formal model decomposition and collaborative development is one of the main obstacles to the cost-efficient deployment of formal verification in industry. In case of Event-B, the formal decomposition approaches are founded on the principle of top-down development. This enables rigorous proof of property preservation through the refinement process. Unfortunately, the way decomposition in Event-B works does not ensure that the code produced for decomposed parts will work efficiently, and that it will be at the same level of details if the decomposition is made at earlier steps of design. We believe

it is necessary to step aside from the formalities of decomposition and explore how decomposition may be efficiently integrated into the development process. To this end, we propose a decomposition method based on a parallel event composition (over disjoint state variable lists) and event fusion (over intersecting state variable lists)[16]. With this method, there is no automatic re-composability guarantee and modellers may disregard formal correctness conditions of model decomposition. In place of the re-composability proof, we offer a protocol that guarantees eventual re-composability and tolerates temporary re-composability conflicts. This method will be naturally supported by our cloud-based architecture: the protocol will be operating over the artefact repository that will ensure that all relevant artefacts are kept and shared among the group of developers. As this first step of this work we have already created an artefact repository as a web service in which the models elements are extracted and saved uniquely with the access permissions granted to the group of stakeholders.

**Dependability:** With the movement of software engineering from local computers to the cloud, software developers need to be assured of the dependability of the engineering support deployed to the cloud. The contracts provided by cloud service providers attempt to assure developers of the cloud performance, availability and reliability. However, due to their inherent complexity and large scale, production cloud computing systems are prone to various run-time problems caused by hardware and software faults, cloud run-time management decisions and environmental factors [7]. System developers and the cloud service providers need to be informed about possible violations of the contracts (when for instance, there is a request *timeout* due to the developer specifying a longer timeout than the cloud's contract provides). We will introduce both modular and system level dependability components that will be implemented to measure in run-time the metrics [6] that are related to the architecture performance, reliability and availability. The data gathered will be compared with those from the cloud platform and the contracts, then a notification will be triggered to the developer(s) and the cloud service provider.

# 6. REFERENCES


[1] B. Aichernig, K. Hormaier, F. Lorber, D. Nickovic, R. Schlick, D. Simoneau, and S. Tiran. Integration of Requirements Engineering and Test-Case Generation via OSLC. In *Quality Software (QSIC), 2014 14th International Conference on*, pages 117–126, Oct 2014.

[2] Amazon Web Service. Amazon Relational Database Service (Amazon RDS), 2014, "Online at http://aws.amazon.com/rds/".

[3] Codenvy Inc. Codenvy Developer Environment Cloud, 2014, "Online at https://codenvy.com/products".

[4] Event-B. Event-B and the Rodin Platform, 2014, "Online at http://www.event-b.org/".

[5] François Bobot and Jean-Christophe Filliâtre and Claude Marché and Andrei Paskevich. Why3: Shepherd your Herd of Provers. In *Workshop on Intermediate Verification Languages*, 2011.

[6] Q. Guan, C.-C. Chiu, and S. Fu. CDA: A Cloud Dependability Analysis Framework for Characterizing System Dependability in Cloud Computing Infrastructures. In *Dependable Computing (PRDC), 2012 IEEE 18th Pacific Rim International Symposium on*, pages 11–20, Nov 2012.

[7] Q. Guan and S. Fu. Adaptive Anomaly Identification by Exploring Metric Subspace in Cloud Computing Infrastructures. In *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*, pages 205–214, Sept 2013.

[8] M. Hapke, A. Jaszkiewicz, K. Kowalczykiewicz, D. Weiss, and P. Zielniewicz. OPHELIA: Open Platform for Distributed Software Development. *Proceedings of Open Source International Conference, Malaga, Spain, Open Source for an Information and knowledge society*, 2004.

[9] Heinrich-Heine-University. The ProB Animator and Model Checker, 2014, "Online at http://www.stups.uni-duesseldorf.de/ProB/index.php5/Main_Page".

[10] A. Iliasov, I. Lopatkin, and A. Romanovsky. The SafeCap Project on Railway Safety Verification and Capacity Simulation. In *Software Engineering for Resilient Systems*, volume 8166, pages 125–132. Springer Berlin Heidelberg, 2013.

[11] Y. Jadeja and K. Modi. Cloud computing - concepts, architecture and challenges. In *Computing, Electronics and Electrical Technologies (ICCEET), 2012 International Conference on*, pages 877–880, March 2012.

[12] Y. Liu and B. Plale. Survey of Publish Subscribe Event Systems, May 2003. In *Technical Report TR574, Indiana University*, "Online at citeseer.ist.psu.edu/liu03survey.html".

[13] Microsoft Research. Z3 - Efficient Theorem Prover, 2014, "Online at http://rise4fun.com/Z3".

[14] A. Pakhira and P. Andras. Using network analysis metrics to discover functionally important methods in large-scale software systems. In *Emerging Trends in Software Metrics (WETSoM), 2012 3rd International Workshop on*, pages 70–76, June 2012.

[15] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252, May 2013.

[16] M. Poppleton. The Composition of Event-B Models. In *ASM, B and Z: First International Conference (ABZ 2008)* on, pages 209–222. September 2008, "Online at http://eprints.soton.ac.uk/266487/".

[17] J. Rushby. An Evidential Tool Bus. In *Proceedings of ICFEM 2005*, 2005.

[18] The Eclipse Foundation. Orion: Open Source Platform For Cloud Based Development, 2014, "Online at http://eclipse.org/orion/".

[19] The Eclipse Foundation. ProR: Requirements Engineering Platform, 2015, "Online at http://eclipse.org/rmf/pror/".

[20] L. Wu, G. Liang, S. Kui, and Q. Wang. CEclipse: An Online IDE for Programing in the Cloud. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 45–52, July 2011.